Application Placement and Infrastructure Optimisation

Radhika Loomba and Keith A. Ellis

Abstract This chapter introduces the RECAP Infrastructure Optimiser tasked with optimal application placement and infrastructure optimisation. The chapter details the methodology, models, and algorithmic approach taken to augment the RECAP Application Optimiser output in producing a more holistic optimisation, cognisant of both application and infrastructure provider interests.

Keywords Resource provisioning • Application placement • Infrastructure optimisation • Infrastructure modelling • Distributed clouds

R. Loomba • K. A. Ellis (✉)
Intel Labs Europe, Dublin, Ireland
e-mail: r.loomba@ieee.org; keith.ellis5@mail.dcu.ie

4.1 INTRODUCTION

As discussed in Chap. 3, the RECAP Application Optimiser derives initial placements from network topologies. These placements utilise application, workload, and workload prediction models to derive scaling models, which are combined with Machine Learning (ML) tools to produce application models and recommendations. So, one might think “given I have an application placement recommendation, what does the application placement and infrastructure optimiser do?”

Well, put simply, the application optimiser does not have all the requisite information to make an optimal decision. When considering an application placement decision, one must take into account that typical operations perceive a separation of concerns between an application provider and an infrastructure provider, the latter typically dealing with multiple application provider requests.

In essence, the RECAP Infrastructure Optimiser brings the interests and heuristics of the infrastructure provider to bear on the overall placement decision and is tasked with three functions:

1. Application Placement: the mapping of individual application service components to individual infrastructure resources. The focus being to identify, traverse, and optimally select from possible placements.
2. Infrastructure Optimisation: focuses on optimising the availability and distribution of an optimal number, type, and configuration of physical resources, while optimising their utilisation, i.e. “sweating the assets”.
3. Capacity Planning: considers future workloads and decides what type of physical resource/node should be placed in the network, how many nodes, and where to place them.

The remainder of this chapter is structured as follows. Section 4.2 briefly outlines the RECAP Infrastructure Optimiser architecture followed by the problem formulation for making a more holistic placement optimisation decision. We then present and discuss three different models used in the RECAP Infrastructure Optimiser, namely load distribution models, infrastructure contextualisation models, and load translation models. Finally, we outline the RECAP algorithmic approach for optimal placement selection.

4.2 HIGH-LEVEL ARCHITECTURE OF THE INFRASTRUCTURE OPTIMISER

Functionally, the RECAP Infrastructure Optimiser as presented in Loomba et al. (2019), can be considered in terms of an “offline infrastructure optimisation modelling” process, Fig. 4.1, and an “online infrastructure optimiser” implementation, Fig. 4.2.

The offline infrastructure optimisation modelling process and the online implementation components/steps are illustrated within the grey

Fig. 4.1 Offline infrastructure optimisation modelling process

Fig. 4.2 Online application placement and infrastructure optimiser

boxes in Figs. 4.1 and 4.2, although it should be noted that infrastructure optimisation is highly dependent on the veracity of its inputs as depicted by the leftmost boxes in Figs. 4.1 and 4.2.

The main steps in the RECAP infrastructure optimisation process are outlined in Table 4.1 below.

4.3 PROBLEM FORMULATION

The application placement and infrastructure optimisation challenge is threefold, i.e. how to:

- Optimally match the requirements of components, e.g. VMs to physical resources
- Adhere to SLAs negotiated between the infrastructure and application providers
- And when to instantiate the virtual components relative to the required capacity

Each challenge is composed of several subchallenges, including but not limited to determining the optimal abstraction of components and resources, defining the objectives that bound the placement, and identifying a common means to compare deployment solutions. This may, in fact, involve multiple-competing objectives; hence, there must be a trade-off.

Current literature suggests stochastic bin-packing method (Jin et al. 2012), approximation algorithms (Meng et al. 2010), and a variety of other heuristics (Li et al. 2014) with focus on specific resources or objectives, e.g. resource usage, and power and thermal dissipation (Xu and Fortes 2010). However, often commercial and open source orchestration solutions schedule either pessimistically to avoid conflicts or opportunistically to gain from potential Total Cost of Ownership (TCO) benefits.

4.3.1 *Infrastructure*

Deploying applications or component instances as VMs or application containers, requires a rich understanding of the heterogeneity and state of the underlying infrastructure. This is mainly because the application workloads might be computation-, storage-, or network intensive. With respect to infrastructure, the requisite information is represented as a multi-layered graph of the physical and logical topology called a “landscape”

Table 4.1 Steps in the RECAP infrastructure optimisation process

<i>Step No.</i>	<i>Step title</i>	<i>Description</i>
1	Input collection	<p>Manages the data ingress of inputs and is essentially the same for offline modelling and the online implementation; what changes is the context. Inputs include:</p> <ul style="list-style-type: none"> • Composition and structure of infrastructure available, i.e. landscape • Composition and structure of the application (e.g. application model and load distributions) • Associated telemetry data, e.g. from testbed or system of interest • Infrastructure provider and service/application provider KPIs
2	Modelling and/or selection	<p>Creates/utilises models primarily in the offline mode to produce outputs that are subsequently codified within the online optimiser. (2a) Combines telemetry with an infrastructure landscape and filters as appropriate based on relevant KPIs. The process is the same for offline and online, only the context changes. Offline (2b) creates application load translation models, which map how application load correlates to resources, associated telemetry and/or KPIs. Online this step more simply involves appropriate selection of a subset of “application load to physical capacity mappings” from those modelled offline. Offline (2c) utilises KPIs and Multi-Attribute Utility Theory (MAUT) to formulate Utility Functions for application and infrastructure providers. Online this step selects a subset of utility functions from those previously formulated offline.</p>
3	Modelling/selection output	<p>Offline (3a) represents a consolidated infrastructure model, i.e. a testbed specific landscape that feeds the “load translation modelling” process. Online this is the use-case-specific landscape and is fed directly to the algorithm module step 4. Offline (3b) encompasses the complete set of possible “application load to physical capacity mappings” based on the testbed inputs. Online it is an appropriate subset of those modelled offline based on the use case inputs. Offline (3c) is the complete set of possible “Utility Functions”. Online it is an appropriate subset based on the given use case.</p>
4	Algorithmic optimisation	<p>This step is illustrated in Fig. 4.2 and takes (3a) and (3b) as inputs. The algorithm subsequently provides several valid solutions, over which the utility functions selected in step (3c) are applied to select a near-optimal option. The output of step 4 is a real-time application placement or a future infrastructure optimisation.</p>

(Metsch et al. 2015). This landscape is built utilising the Neo4J database. It is primarily a graph describing a computing infrastructure, that also details what software stacks are running on what virtual infrastructure, and what virtual infrastructure is running on what physical infrastructure. The data within a landscape is collected via collectors and event listeners. Collectors are provided for physical hardware information (via the `hwloc` and `cpufreq` files) and for OpenStack environments.

This rich representation helps to understand the capability of the infrastructure. It is mathematically quantified as a landscape graph $G = (N^*, E^*)$ where N^* is the superset of all nodes of all geographical sites indicating resources such as CPU cores, processing units, and switches and E^* is the superset of all links between them, which might be communication links or links showing a relationship between two nodes.

Although this granularity of information is required, it increases the complexity of problem in terms of possible placement combinations and adds additional dimensions. For example, instead of determining at the aggregate server level, one must determine the cores, the processor bus, and the processing units involved in the mapping. As such, for simplicity, the landscape graph is abstracted in this initial formulation into a contextualised landscape graph $G_1 = (X, E)$ where $X \subset N^*$ and set $E \subset E^*$ containing only two categories of node, namely network and compute. Set X is a collection of nodes with a compute or network category and Set E is the set of all links connecting these nodes. This abstraction defines a network node to be of type, e.g. `nic`, `vnic`, `switch`, `router` or `SRIOV channel`. A compute node is defined as a resource aggregate of a given type, e.g. `machine` or `virtual machine` and is created by collapsing the properties of the individual compute, memory, and storage entities directly connected and contained within the node. This helps isolate the two categories of nodes while storing pertinent information regarding the other categories.

Building on work and experience of the Superfluidity project (Superfluidity Consortium 2017), these nodes also contain attributes which quantify their capacity. This is represented in a vector format as $\mathbf{v}_x$ for node x along with telemetry information regarding utilisation (the average percentage of times a resource is deemed to be “busy” over a pre-defined time window for the given resource), saturation (the average percentage of times a resource has been assigned more tasks than it can currently complete), cost models, etc. The superscripts c , m , n , and s denote compute, memory, network, and storage category values respectively. The compute node ς has capacity $\mathbf{v}_\varsigma = [\mathbf{v}_\varsigma^c, \mathbf{v}_\varsigma^m, \mathbf{v}_\varsigma^s]$ where $\mathbf{v}_\varsigma^c$

represents the number of cores and v_x^m the amount of free or unused RAM from the total installed on the resource aggregate, and v_x^s represents disk size. A network node $\mathbf{n}$ has a 2-tuple capacity vector $\mathbf{v}_x = [v_x'', v_x^*]$ calculated based on its available bandwidth v'' and available connections v^* .

Furthermore, for physical communication link $e \in E$ representing the graph edge, link attributes are added including geographical distance $len(e)$ and measured or observed throughput τ_e and latency l_e normalised for time δt , just before any application placement decision is made. Values such as B_e for maximum bandwidth and associated rate R_e^b for a unit of bandwidth are also included in the infrastructure graph G_1 .

4.3.2 Application

The application to be deployed is itself described as a service request, composed of either multiple connected service components in the form of service function chains (SFC), or disjointed service components, which need to be placed together. Using the definitions presented in (RECAP Consortium 2018), the application is represented in a service request graph $G_2 = (Z, F)$ with nodes represented by set Z and graph edges by set F . For this model, the nodes are further categorised similar to the method described above into either compute or network nodes and are termed as request nodes and request graph edges as they represent the service request.

4.3.3 Mapping Applications to Infrastructure

Each service component of the service request graph G_2 is then mapped on to infrastructure resources and links in graph G_1 . This mapping is composed of a subset of nodes and graph edges of graph G_1 , and as such it is important to first define the rules of such a mapping. This is represented in Fig. 4.3.

The nodes in graph G_2 are defined as a 1:1 mapping with resource nodes, defined by the set of infrastructure nodes $\mathcal{Y}$. This set $\mathcal{Y} \subset X$ contains all compute and network nodes which have a service component mapped to them. This is also quantified as $z \rightarrow y$ where request node $z \in Z$ is mapped to resource node $y \in \mathcal{Y}$, also ensuring that $Z \cong \mathcal{Y}$. There is also a 1:N mapping for the graph edges as they get mapped to a set of

Fig. 4.3 Mapping a service request graph and a contextualised landscape graph

communication links also indicating a physical network path. This is defined as graph edge $f \in F$ being mapped to the set of communication links or path g , which is a subset of all possible paths in the infrastructure graph, also quantified as $f \rightarrow g$. These definitions describe a possible placement solution, denoted by $G_2 \rightarrow G_1$, which may or may not fulfil the criterion of an optimal mapping.

4.3.4 Mapping Constraints

The RECAP Infrastructure Optimiser's objective is to identify an optimal mapping, and this requires the analysis of several constraints. A subset of the most relevant constraints to be considered for an infrastructure resource request model along with its associated policies is presented here. These include the capacity requirement, compositional, SLA, and infrastructure-policies constraints.

4.3.4.1 Capacity Requirement Constraint

This constraint defines the capacity, specified in terms of compute, memory, network and/or storage capacity that must be available on the resources and edges that are intrinsically part of the mapping. For the initial formulation, this capacity is considered static during the duration of the application deployment.

For compute, this includes request information from the application for customised virtual compute flavours, related to the software image and the number of requested cores. Additionally, details on acceleration capabilities, CPU architecture, or clock frequency requests may also be included. For memory, this specifies the amount of virtual memory that needs to be allocated to the request node, and also includes details on whether Non-Uniform Memory Access (NUMA) support is required. For network, this includes request information on the required network bandwidth, the requested network interface (e.g. normal virtual NIC vs. SR-IOV) and additional acceleration capabilities. For storage, this includes request information on the required storage size, type of storage (e.g. SSD vs. HDD), and redundancy level (i.e. RAID).

In this scenario, it is imperative to remember that since the request edge is mapped to a set of communication links or a physical path, the aggregated bandwidth and aggregated latency of all edges that are a part of this physical path must meet the requirement. There are a number of reasons why this requirement must be met including propagation delay, serialisation, data protocols, routing and switching, and queuing and buffering. Of these, the most significant ones are the propagation delays and the queuing delays. Since the network devices are considered in the model, the saturation value of the nodes on the physical path are summed to get the queuing latency, and the aggregation of the bandwidth and propagation latency are quantified as follows.

- The aggregated bandwidth of the physical path is the minimum observed throughput of all edges in the physical path. Mathematically, $b_g = \min_{e \in g} \tau_e$
- The aggregated latency of the physical path is the summation of the propagation latency in the physical path. Mathematically, $l_g = \sum_{e \in g} l_e$

4.3.4.2 Compositional Constraint

Compositional constraint defines any rules that explicitly dictate the composition of the mapping at different levels of granularity such as resource

types (e.g. compute and network), resource groups (e.g. a set of compute resources), or resource zones (e.g. a set of machines deployed at a particular location). At each level of granularity, the constraint can be further quantified as being affinity (“same” or “share”) or anti-affinity specific (“not-same” or “not-share”). These dictates whether resources “share” the “same” physical resource, set of resources, properties or zones, or not.

The first example of such a constraint is the resource type mapping constraint; e.g. a virtual CPU (vCPU) must be mapped to a CPU or a virtual network interface (vNIC) must be mapped to a port of physical network interface (NIC). These are necessary since the network interface to which the vNIC is mapped needs to be on the same server as the CPU host, whereas depending on the configuration of the infrastructure and level of redundancy, the virtual storage component or disk can be mapped to a remote storage disk or to the local storage. Additionally, physical memory banks connected to the physical core allocated to the request deliver different performance in comparison to allocating memory banks connected to other physical cores.

4.3.4.3 *Service Level Agreement Constraint*

This constraint relates to specific customer service requirements and covers information regarding the scheduling prioritisation of individual service customers and application instances, as well as control policies to be enforced to pre-emptively suspend/kill other currently deployed services by the same or a different service customer.

This is modelled by defining a set of SLAs denoted by set S , negotiated between the infrastructure provider and the service provider/customer. The infrastructure provider creates and offers more than one SLA template. These are arranged in a hierarchical manner, with each level relating to a specific “type” of SLA such as “platinum”, “gold” or “silver”. Each template $s \in S$ is associated with one type that defines its rate R_s , the threshold of service level KPIs required by the customer for that template, the unit cost of SLA violations $j \in J_s$ for each KPI, and the list of failure-tolerant implementations $h \in H$ that need to be made by the resource provider.

The customer selects the setting they want within the chosen SLA template and can select more than one SLA for different applications. This becomes the agreed SLA for the application for the customer and is included as a customer request. As such, it also includes the total run-time (in hours) for the application instance T_r and other constraints that the customer requests.

4.3.4.4 *Infrastructure-Policies Constraint*

Apart from scheduling prioritisation, it is also important for the resource provider to define policies and control protocols for the management of the infrastructure. These constraints include resource allocation prioritisation and allocation ratios.

The resource provider can prioritise certain resource types or groups, based on either their cost or performance. These resource groups can also be associated with a particular SLA template and with allocation restricted in certain situations. Also, policies related to overprovisioning of resources need to be defined by the resource provider. This controls the ratio of allocating virtual resources to the physical resource and may differ by category of resource and specific use case. Additionally, it includes the minimum and maximum capacity that is allocated to one instance over the entire run-time of the application instance, if any.

4.4 MODELS THAT INFORM INFRASTRUCTURE OPTIMISATION DECISIONS

As discussed earlier, three models are used in the RECAP Optimisation Framework to inform optimisation decisions—load distribution models, infrastructure contextualisation models, and load translation models. As load distribution models have been discussed earlier in Chap. 3, we will focus on infrastructure contextualisation models and load translation models here.

4.4.1 *Infrastructure contextualisation models*

While “the map is seldom the territory”, a good map invariably helps. As discussed earlier, the “infrastructure representation (landscape)” is an important input to the RECAP Infrastructure Optimiser and aims to provide a rich representation of the resource composition, configuration, and topology of the various entities in the cloud/edge infrastructure, across three layers—physical, virtual, and service. However, a landscape for a given scenario may not have all the requisite data expected (e.g. geographical and capacity), or it may be too rich having redundant information irrelevant to the specific use case. Additionally, it will not have telemetry data needed to support the optimisation process, e.g. current utilisation. Furthermore, if granularity of the infrastructural information is increased,

multiple different mappings need to be considered, increasing the complexity of the NP-hard problem.

To address this issue, a “contextualised modelling process” (Figs. 4.1(2a)/4.2(2a)) is undertaken so as to produce a “consolidated infrastructure model” (Figs. 4.1(3a) and 4.2(3a)). This process may augment and/or filter the landscape input, adding telemetry, e.g. resource utilisation, KPIs (perhaps to filter), and important platform features identified for the individual use cases. But assuming the landscape provided meets the requirements of the optimiser, the only addition for creation of the consolidated model is telemetry, plus any required filtering based on appropriate KPIs. The type of information encapsulated in the consolidated model is illustrated by way of the following brief example. Figure 4.4 represents the network topology for the city of Umeå, Sweden.

The contextualised infrastructure model for such a network includes:

Fig. 4.4 Overview of Umeå network topology (site routers only)

- The definition of resource sites (e.g. MSAN and Metro), i.e. information pertaining to individual physical infrastructure elements, their physical attributes, and configurations, the communication links between them, and the properties inherent to these links.
- The definition of inter-site network bandwidth and latency, i.e. the available network bandwidth and latency values of the physical communication links between resource sites.

The output of this process is a representative graph of the network landscape. At a more granular level, Fig. 4.5 below shows the modelled communication links across the tiers for just one resource site T21, as presented by the Neo4J database.

4.4.2 Load Translation Models

Given a good understanding of the physical infrastructure, one must then consider the applications that are to be optimally deployed. In that regard and building on Chap. 3, load translation models serve to:

1. Quantify the association between virtual machine/container configurations and specific infrastructure configurations, and
2. Determine the lower and upper bounds on resource consumption in relation to varied application performance KPIs.

The RECAP load translation models are designed to be generally applicable to distributed application deployments and cloud/edge

Fig. 4.5 Representation of a subgraph of contextualised network landscape

infrastructures. The methodology presented here focuses on mapping the virtual workload of individual service components to a set of prioritised time series data-points, i.e. telemetry collected from the physical infrastructure components. The correlation of service components and telemetry informs an offline modelling process that builds insights about application placement and related performance of deployed components. These insights which are then available to the online optimiser.

RECAP experimentation highlighted how understanding the profile mix of infrastructure features coupled with the different service components, e.g. Virtual Network Functions (VNFs) deployed on the infrastructure impacts the efficient usage and distribution of physical resources, i.e. compute, memory, network, and storage.

The RECAP load translation methodology is illustrated in Fig. 4.6.

The methodology begins with understanding the following inputs:

- **Infrastructure Information:** It is important to understand the testbed/experimentation set-up and to understand its limitations, especially those relating to the heterogeneity of available server configurations. This helps define the information on the infrastructure, e.g. physical machine, virtual machine and container configurations, total number of machines, and their connections.
- **Associated Telemetry Data:** A telemetry agent is initialised to collect data over multiple domains, e.g. compute, network, storage, and memory. All incoming data from these metrics are aggregated and normalised based on the domain context and studied as time series data.
- **Virtual Machine Configurations:** The application to be placed on these machines needs to be understood, how many instances and types of instances can be run, whether they are deployed as service-chain or disjoint VMs/containers, along with the various configurations of these VMs/containers. This information is typically provided via the application optimiser.
- **End-User Metadata:** The end-user behaviour to be emulated is determined based upon use case definitions and validation scenarios, and this specifies how users will access the applications, how many users will be initialised, how the number of users will increase/decrease, and how different user behaviours will be simulated in the testbed.

Fig. 4.6 The load translation methodology in full

These four inputs are then used to define the context of experiments that have to be run which are based on, for example, the duration of each experimental run; the prioritisation of configurations; application instances and end-user workloads that will be varied; and the number, type, and behaviour of users that will be applied for each experiment, as well as the number of times the same experimental set-up will be replicated for redundancy purposes. This helps define a set of profiles that are given to the load translation model to assess and analyse.

Once the experimental data is available and the experiment defined, the following data analysis steps are undertaken:

- **Data Wrangling:** The collected data is isolated and labelled appropriately according to experimentally relevant timestamps.
- **Data Filtering:** Files are filtered and integrated as appropriate for comparison.
- **Data Visualisation and Analysis:** Visualisation and analysis are completed for each defined experiment. The appropriate metrics are defined and calculated per machine/VM/container as well as device names, e.g. for network interfaces and storage devices attached to the physical machine. The results are then summarised and are visualised as a comparison between the average usage for compute, memory, network, and storage resources.

For illustration purposes, Fig. 4.7 shows compute utilisation for two machines (compute-1 and compute-3) for the different VNF placement profiles of Use-Case A, normalised given the number of cores of each machine. This type of analysis provides an initial and basic understanding of the relation between the workload, application, and infrastructure. But as more and more data gets collected and analysed, the accuracy of the normalisation across experiments, quantification of the relationship, and formulation of the mathematical equation for same increases. The results of this process are collated as a complete set of mappings to be used for specific use case optimisations.

Completing this may still seem relatively straightforward until one considers the different KPIs, constraints, complexity, and scale to be addressed.

Fig. 4.7 Average compute utilisation by VNF placement profile for compute-1 compute-3

Given such scale and complexity, understanding, traversing, and promptly selecting from such nuanced options at scale must be mathematically derived and programmatically implemented, and this is the task of the algorithmic approach discussed next.

4.5 ALGORITHMIC APPROACH TO OPTIMAL SELECTION

It should be apparent that application placement and infrastructure optimisation is highly dependent on the veracity of inputs received and that the optimiser is a collective of components and models coupled with the algorithmic approach applied to the output of those models, not just an algorithm. As such, this section describes (1) the utility functions, and (2) the evolutionary algorithm used in the RECAP Infrastructure Optimiser. The former is used as a uniform mathematical framework to normalise business objectives to compare possible placements identified by the algorithm. The latter was chosen for its appropriateness in quickly identifying and selecting near-optimal placement options.

4.5.1 *Utility Functions*

Previously the application placement problem was defined as optimally matching service requests to the capabilities of available resources, instantiating these components with the required capacity, and finally meeting SLAs between the resource and application providers. This transforms the problem into iteratively mapping individual service components on to various available infrastructural resources while meeting the constraints defined above. Solving this, presents many possible placement solutions out of which the optimal solution needs to be selected. Moreover, the number of possible solutions increases with growing sizes of application/service requests and infrastructure, further increasing the complexity.

Determining the optimal solution is thus intrinsically challenging as it entails comparing deployments based on their benefits to either the provider or the service customer over a large solution space in both time and space. Furthermore, distinct yet complementary objectives and constraints must be handled, and trade-offs made. These objectives are often in different scales, ranges, and units, and need to be normalised into a common reference space.

Enter “Utility Functions”, a key mechanism enabling analysis across varied objectives for different placement options and focused on

Fig. 4.8 Provider-centric vs customer-centric attributes (Loomba et al. 2017)

understanding the “reward” that is acquired per objective. Loomba et al. (2017) quantified these benefits as two utility functions, one for the resource provider denoted as U_p and one for the service customer denoted as U_c . These objectives are classified as being Provider-centric and Customer-centric, in Fig. 4.8.

1. **Provider-centric Objectives:** These objectives relate to the resource provider and deal with the management of the entire infrastructure. Key objectives include:
 - **Gross Profit:** This objective includes calculated revenue and expenditure costs for resource capacity allocated, cost of associated SLAs and SLA violations, and other costs of concern to the infrastructure provider.

- Service Distribution: This objective includes the analysis of available capacity of resources after the application deployment.
2. **Customer-centric Objectives:** These objectives relate to the service customer, quantifying Quality of Experience (QoE) as well as Quality-of-Service (QoS). These also help reason over the goodness of a placement decision and key objectives include:
- Throughput: This objective includes the quantification of observed throughput over all physical links in the physical network, along with the analysis of dropped packets.
 - Latency: This objective includes the quantification of observed latency over all edges in the physical network, along with the analysis of packet delays.

This approach is extended in the RECAP methodology to consider enhanced constraints and use case definitions in defining a combined utility function that negotiates trade-offs between these two utility functions. This combined utility incorporated the preferences and priorities of the various use case business objectives and was evaluated using the Multi-Attribute Utility Theory (MAUT).

In this formulation, a multiplicative function is used to capture the interdependence of k conditional utilities for each attribute $a_i \in \mathcal{A}$. Here $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2 = \{a_1 \dots a_i \dots a_k\}$, with $k \geq 2$ is the set of all objectives under consideration with subset $\mathcal{A}_1$ containing all provider-centric objectives and subset $\mathcal{A}_2$ containing all customer-centric objectives. For each of these conditional utilities, α_k indicates weight or a priority value of the objective while β_k is an additive weight that stores dependence on other objectives.

$$U_p = f(\alpha_i \cdot \mathcal{U}(a_i) + \beta_i), \text{ where } a_i \in \mathcal{A}_1$$

$$U_c = f(\alpha_i \cdot \mathcal{U}(a_i) + \beta_i), \text{ where } a_i \in \mathcal{A}_2$$

By assigning prioritisation weights to the provider utility and customer utility, the total utility of the placement can be calculated. These weights must be modifiable as they bias the selection of the placement solution. The total utility of the deployment with w_1 weight to the provider utility and w_2 weight to the customer utility can thus be defined as a weighted summation of the inputs. The optimisation function is then defined to maximise the total utility of the placement, considering minimum and

maximum thresholds for the provider utility and customer utility. This is designed to facilitate graceful expansion to accommodate any variables/parameters outside the scope of the current formulation or that gain importance to the use case owner following this analysis. The optimisation function is represented as follows:

$$\text{Maximise : } w_1 U_p + w_2 U_c$$

Subject to:

$$1. \quad \min_{\text{threshold}} (U_p) \leq U_p \leq \max_{\text{threshold}} (U_p)$$

$$2. \quad \min_{\text{threshold}} (U_c) \leq U_c \leq \max_{\text{threshold}} (U_c)$$

These use case specific utility functions thus ensure that the optimiser can adapt and reason over placement decisions even when business objective weights or priorities are changed.

4.5.2 *Algorithms for Infrastructure Optimisation and Application Placement*

While utility functions provide a mathematical framework for comparison, an initial challenge exists in being able to quickly select a subset of optimal solutions from the large number of possible deployment solutions for comparison.

In addressing this challenge, a *stochastic evolutionary algorithm* was selected. Its appropriateness relates to its incorporation of enough randomness and control to support decision making even for functions lacking continuity, derivatives, linearity, or other features.

Additionally, its ability to exploit historical information makes the algorithm much more efficient and powerful in comparison to exhaustive and

random search techniques. Its advantages further include the ability to isolate a set of “good” solutions instead of just one, the possibility of parallelisation to improve efficiency and the support for multi-objective problems.

The algorithm calculates the optimal solution(s) in an admissible region for this combinatorically complex problem, which otherwise could not be solved by polynomial time. The optimality of the solution is based on its quality criterion called the “fitness function” and is represented as $f_{G_2 \rightarrow G_1}$ for the deployment solution $G_2 \rightarrow G_1$. This value is composed of the fitness of the individual mapping, based on the constraint definitions presented above. In the given application placement scenario, the objective of the algorithm is $\min f_{G_2 \rightarrow G_1}, \forall x \in [G_2 \rightarrow G_1]$ to ensure fast convergence for solutions that do not meet the required constraints. The output of the algorithm is thus the placement solution (or set of placement solutions) with minimum fitness or with fitness tending to zero (whichever is lower).

An overview of the algorithm is outlined in Fig. 4.9 and the accompanying text:

The set of possible placement solutions represents a “population”, where each of these solutions is a “candidate” for the algorithm. The individual mappings in the placement solution are treated as “genes” (e.g. the mapping of a service component to an infrastructure resource).

Fig. 4.9 Representation of the evolutionary algorithm

Initialisation: The algorithm is defined by a set of placement solutions or a set of candidates, which defines the current “population”. These candidates can be arbitrarily chosen or can be based on prior knowledge from the use case to encompass a wide range of solutions. This is essential as with many different mappings from the placement solutions or “genes” present, it becomes easier to explore several different possibilities during the algorithm.

Evaluation: Each candidate in the population is evaluated according to the defined fitness function which numerically represents the viability of the solution. Thus, the next step is to eliminate the portion of the population with worse fitness values. The population now contains fitter “genes” or those mappings that fulfil defined placement criterion and have low fitness values.

Evolution: This step involves two main operations, mutation and cross-over. Both these operations are probabilistic and are used to create the next population from the remaining candidates.

- Mutation is used to introduce new candidates in the population by spontaneously changing one gene. In RECAP, this means that multiple mappings within a possible placement solution are swapped to create a new possible placement solution.
- Cross-over creates a mix of the available genes in the population by merging the genes from two candidates. In RECAP, this means that two mappings of different service components are taken from two different possible placement solutions and combined to create a new possible placement.

In either case, the resulting candidate may have better or worse fitness than existing members.

Termination: The algorithm ends in two cases. First after 30 iterations of the algorithm, for statistical accuracy and the second when it has found deployment solutions with fitness value equal to zero. All identified solutions are returned and evaluated and the solution with highest total utility is selected.

4.6 CONCLUSION

The RECAP Infrastructure Optimiser and the RECAP Application Optimiser are interdependent components in the RECAP ecosystem. The RECAP Application Optimiser does not have all the requisite information for an optimal decision, especially when considering that the typical mode of operation perceives a separation of concerns between the application and infrastructure providers. Therefore, the RECAP Infrastructure Optimiser adds value by augmenting the application optimisation with additional infrastructure-specific information that can encompass the business objectives of both application and infrastructure providers.

In this chapter, we detailed the infrastructure optimisation architecture tasked with establishing this more holistic optimisation recommendation. Section 4.3 outlined the problem formulation describing the varied and detailed inputs required for optimisation that must be mathematically and programmatically described and traversed, including infrastructural components, the application, and constraints to optimisation. This was followed by a description of the models that inform optimisation, and the evolutionary algorithm and utility functions used to mathematically and programmatically select from sub-optimal solutions.

The value in the described approach is difficult to articulate and visualise. When one considers the size and complexity of modern hyperscale architecture, it is apparent that such a granular, mathematical, and programmatically implementable approach is required in order to extract value from the nuanced and humanly incomprehensible myriad of available options.

REFERENCES

- Jin, Hao, Deng Pan, Jing Xu, and N. Pissinou. 2012. *Efficient VM Placement with Multiple Deterministic and Stochastic Resources in Data Centers*. Proceedings of the IEEE Global Communications Conference (GLOBECOM), 2505–2510.
- Li, X., A. Ventresque, J. Murphy, and J. Thorburn. 2014. *A Fair Comparison of VM Placement Heuristics and a More Effective Solution*. Proceedings of the 13th IEEE International Symposium on Parallel and Distributed Computing (ISPDC), 35–42.
- Loomba, R., T. Metsch, L. Feehan, and J. Butler. 2017. *Utility-Driven Deployment Decision Making*. Proceedings of the 10th International Conference on Utility and Cloud Computing, Austin, TX, 207–208.

- Loomba, Radhika, Keith A. Ellis, Paolo Casari, Rafael García, Thang Le Duc, Per-Olov Östberg, and Johan Forsman. 2019. *Final Infrastructure Optimisation and Orchestration*. RECAP Deliverable 8.4, Dublin, Ireland.
- Meng, X., V. Pappas, and L. Zhang. 2010. *Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement*. IEEE INFOCOM, San Diego.
- Metsch, T., O. Ibidunmoye, V. Bayon-Molino, J. Butler, F Hernández-Rodriguez, and E. Elmroth. 2015. *Apex Lake: A Framework for Enabling Smart Orchestration*. Proceedings of the Industrial Track of the 16th International Middleware Conference, Vancouver.
- RECAP Consortium. 2018. Deliverable 6.1. Initial Workload, Load Propagation, and Application Models.
- Superfluidity Consortium. 2017. Deliverable 5.1. Function Allocation Algorithms Implementation and Evaluation.
- Xu, J., and J.A.B. Fortes. 2010. *Multi-objective Virtual Machine Placement in Virtualized Data Center Environments*. Proceedings of IEEE/ACM International Conference on Green Computing and Communications (GreenCom) & International Conference on Cyber, Physical and Social Computing (CPSCom), Hangzhou, 179–188.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.